Стек протоколов BitCloud фирмы Atmel

Часть 1. Основные свойства и структура

Беспроводные сенсорные сети и системы управления постепенно становятся все более привычным явлением. Подавляющее число предлагаемых производителями и разработчиками решений в той или иной мере основывается на стандартах 802.15, ZigBee, ZigBeePRO. Сейчас производители выпускают не просто приемопередатчики, физически поддерживающие требуемые частотные диапазоны, форматы данных, скорости передачи, но и предоставляют готовые стеки протоколов, позволяющие организовывать сети различной топологии, осуществлять маршрутизацию, передачу данных и многое другое. Наличие готового стека протоколов позволяет разработчикам сосредоточить внимание именно на прикладных сетевых приложениях — примерно так, как это происходит с веб-сервисами и клиент-серверными приложениями для персональных компьютеров: стек сетевых протоколов выполняет основную работу.

Александр Калачев

Стек протоколов BitCloud представляет собой полнофункциональный программный стек для встраиваемых систем, позволяющий создавать на его основе надежные, масштабируемые и безопасные приложения, работающие на аппаратной платформе фирмы Atmel. Основные области применений включают в себя автоматизацию дома, коммерческих зданий, автоматические системы измерений и сбора данных, отслеживание положения предметов или транспортных средств, средства промышленной автоматизации… И этим область применений данного программного стека не ограничивается. BitCloud полностью совместим со стандартами беспроводных сенсорных систем и систем управления ZigBee и ZigBeePRO [7, 8]. Программный стек предоставляет разработчикам приложений расширенный набор программных интерфейсов (API), сочетающих в себе соответствие стандартам и возможности быстрой разработки приложений, что существенно сокращает время выхода продукта на рынок.

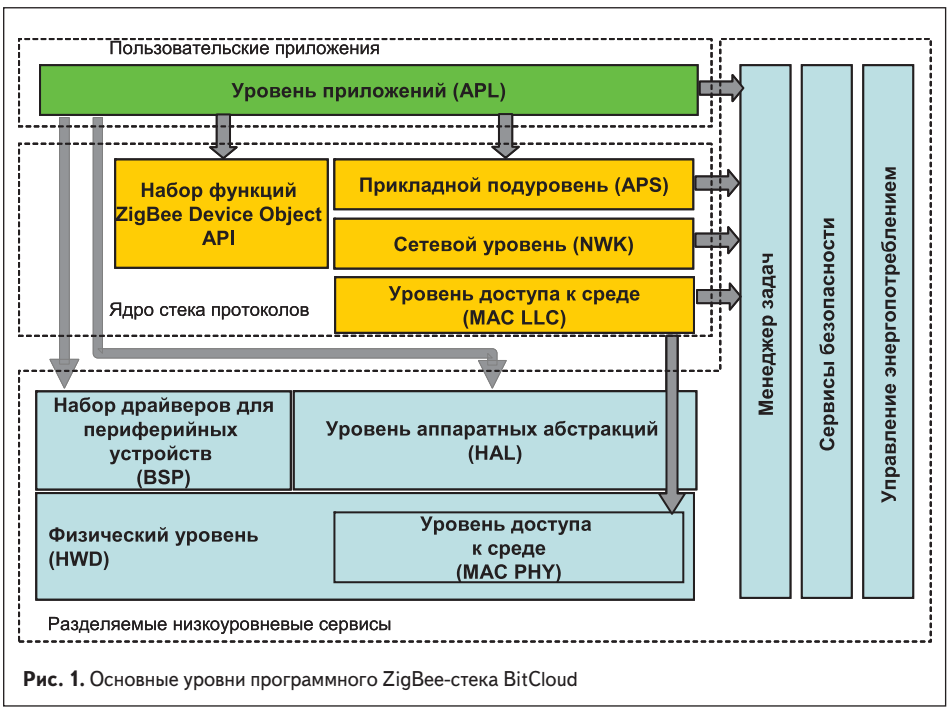

Рис. 1. Основные уровни программного ZigBee-стека BitCloud

Таблица 1. Файловая структура BitCloud SDK

Директория	Исходные тексты	Описание
BitCloud		Корневая директория стека
APS	Не предоставляются	Прикладной подуровень — подуровень поддержки приложений
BSP	Предоставляются	Драйвера для периферийных и встраиваемых устройств наборов разработчика
ConfigServer	Предоставляются	Подуровень хранения основных параметров стека протоколов — размеры сети, адресация и пр.
HAL	Предоставляются	Уровень аппаратных абстракций — драйвера устройств для поддерживаемых платформ
MAC_PHY	Не предоставляются	Уровень контроля доступа к среде, физический
NWK	Не предоставляются	Сетевой уровень
PersistDataServer	Предоставляются	Доступ к данным EEPROM и постоянным параметрам
Security	Не предоставляются	Сервисы обеспечения безопасности
SystemEnvironment	Не предоставляются	Менеджер задач и реализация функции main()
ZDO	Не предоставляются	Подуровень объектов ZigBee-устройств
WSNDemo	Предоставляются	Полнофункциональное сетевое приложение, демонстрирующее сбор данных, обеспечение безопасности и управление питанием узлов сети

Поддерживаемые платформы включают в себя 8- и 32-битные контроллеры Atmel совместно с беспроводными приемопередатчиками, выполненные в виде интегральных модулей поверхностного монтажа [1–5]. Существуют следующие версии BitCloud SDK [6]:

- SDK for ATAVRRZRAVEN — AVRRAVEN и RZUSBSTICK (часть набора ATAVRRZRAVEN);
- SDK for ZigBit — ATZB-DK-24 (ZDK);
- SDK for ZigBit Amp — ATZB-DK-A24 (ZDK Amp);
- SDK for ZigBit 900 — ATZB-DK-900 (ZDK 900).

Общая структура стека протоколов BitCloud

Внутренняя структура BitCloud следует соглашениям по разделению функций сетевого стека протоколов на логические уровни, как это определено в стандартах IEEE 802.15.4 и ZigBee [7–10]. Кроме ядра стека протоколов, BitCloud содержит дополнительные сервисы (менеджер задач, безопасность, управление питанием), уровни аппаратных абстракций (HAL) и пакеты поддержки отладочных комплектов (BSP) (рис. 1). Все эти дополнения существенно облегчают разработку и отладку приложений, а также увеличивают их функциональность.

Выделяют три группы уровней и наборов функций, доступных пользовательским приложениям (APL).

- Аппаратные функции:
  BSP — набор драйверов для управления стандартным комплектом периферийных устройств, расположенных на платах оценочных комплектов (сенсоры, кнопки, светодиоды и пр.);
  HAL — набор функций для доступа к аппаратным ресурсам модуля ZigBit (EEPROM, таймеры) и периферийных устройств управляющего контроллера (внешние прерывания, последовательные интерфейсы — TWI, SPI, UART, 1-wire, управления линиями ввода-вывода, работа с АЦП).
- Сетевые функции:
  ZDO — полностью совместимый со стандартом ZigBee набор функций ZigBee Device Object API для реализации базовой сетевой функциональности — начало работы узла, сброс состояния, создание сети, подключение к существующей сети и т. п. Определяет также тип устройства ZigBee, его профиль, позволяет задать набор команд для управления устройством.
  APS — самый верхний уровень сетевого стека, доступный пользовательскому приложению.

Существует также набор сервисов, доступный приложению, охватывающий все уровни стека протоколов, — так называемые «вертикальные» компоненты стека / «вертикальные» сервисы. В него входят:

- Менеджер задач, управляющий распределением времени работы и ресурсов управляющего процессора между компонентами стека и пользовательскими задачами. Реализован как приоритетная кооперативная многозадачность, адаптированная под требования многоуровневого сетевого стека, с учетом временных ограничений.
- Управление питанием — функции перевода модуля в режим низкого энергопотребления (перевод в спящий режим) с сохранением состояния компонент сетевого стека и восстановления состояния при пробуждении.
- Безопасность — поддержка шифрования сетевого трафика.

Набор инструментальных средств разработки программного обеспечения BitCloud SDK

BitCloud предоставляется как набор инструментальных средств разработки программного обеспечения (BitCloud SDK), включающий библиотеки, заголовочные файлы, файлы справок, документацию, примеры приложений для всего спектра поддерживаемых платформ.

Основные свойства:

- полная совместимость со стандартами ZigBee и ZigBeePRO;
- набор С-функций (С API);
- поддержка АТ-команд;
- поддержка маршрутизации данных для надежной их доставки;
- возможность создания больших сетей (с числом узлов порядка нескольких сотен);
- оптимизация энергопотребления;
- расширенный набор функций обеспечения безопасности данных (поддержка шифрования).

Файловая структура среды разработки (табл. 1) отражает структуру стека протоколов с подпапками, соответствующими структуре отдельных уровней. Компоненты, распространяемые в бинарной форме, содержат директории lib, include, описывающие интерфейс к компоненту, содержащие заголовочные файлы и скомпилированные образы. Компоненты, распространяемые в виде исходных кодов, содержат поддиректории 'objs' и 'src' и файл ***Makefile*** со скриптом, позволяющим скомпилировать компонент [11, 12].

Прикладные приложения всегда содержат заголовочные файлы, файлы исходных текстов, скрипт ***Makefile*** для компиляции приложения и файлы проекта для поддерживаемых сред разработки. Задача ***Makefile*** — скомпилировать приложения, используя заголовочные файлы соответствующих компонентов стека (из соответствующих директорий), и связать полученный код с объектным файлом образа компонента стека. Полученный бинарный образ представляет собой скомпилированное приложение, которое может быть прошито и отлажено в целевом устройстве [11, 12].

Из-за большого количества функций, а также для упрощения программирования имена доступных API-функций начинаются с префикса, обозначающего имя уровня сетевого стека, к которому эта функция относится, отделенного от имени символом «_»:

```
ZDO_GetLqiRssi — относится к уровню ZDO.
```

Функции, начинающиеся на ***Set*** или ***Get***, устанавливают или запрашивают какие-либо параметры от низлежащего уровня:

```
HAL_GetSystemTime.
```

Суффиксы ***Req*** и ***Request*** в имени функции указывают на асинхронные запросы пользовательского приложения сетевому стеку:

```
APS_DataReq.
```

Ind показывает асинхронную индикацию состояния или события, передаваемых пользовательскому приложению от стека:

```
ZDO_NetworkLostInd.
```

По умолчанию имена функций, заканчивающиеся на ***Conf***, являются определяемыми пользователями процедурами обратного вызова для асинхронных запросов, подтверждающих выполнение запрошенных действий.

Каждая структура или тип данных имеют суффикс «***_t***», обозначающий их тип. Перечисления и имена различных макросов обозначены заглавными буквами. Разработчикам приложений также рекомендуется следовать принятой системе обозначений.

Особенности построения приложений с использованием BitCloud

Разработка сетевых приложений с использованием BitCloud ориентирована на управление событиями — стиль проектирования программных систем, при котором поведение компонента системы определяется набором возможных внешних событий и ответных реакций компонента на них. Программно пользовательское приложение предусматривает наличие низлежащих уровней с указателями функций, которые низлежащие слои вызывают при обработке запроса. Что-то аналогичное происходит и в системах с графическим пользовательским интерфейсом (GUI).

Фактически все внутренние интерфейсы стека протоколов определены в терминах заблаговременных запросов и соответствующих им обратных вызовов. Каждый из уровней стека протоколов определяет ряд процедур обратного вызова к низлежащим слоям и, в свою очередь, содержит функции обратного вызова, определенные слоями, лежащими выше.

Запрос является асинхронным вызовом процедуры низлежащего слоя на предмет выполнения каких-либо действий; подтверждение — процедура обратного вызова, выполняющаяся, когда запрошенное действие завершено и становится доступным его результат. Следует обратить внимание на необходимость разнести по времени запрос и ожидание ответа, особенно если выполнение запроса может продолжаться неопределенно долго.

Некоторые системные вызовы, особенно те, что занимают определенное время, являются синхронными. Вызовы одной пользовательской функции из другой также синхронные.

Существуют также случаи, когда приложение пользователя должно быть извещено о внешнем событии, которое не является результатом какого-либо запроса, — например, потеря сети, готовность перехода нижележащих уровней стека в спящее состояние или сигнал о пробуждении системы. Это асинхронные вызовы (рис. 2).

Существует общий тип определяемых пользователем процедур обратного вызова, ответственных за исполнение кода уровня приложения, — ***APL_TaskHandler***, который является зарезервированным именем процедуры обратного вызова, известный уровням стека как менеджер задач.

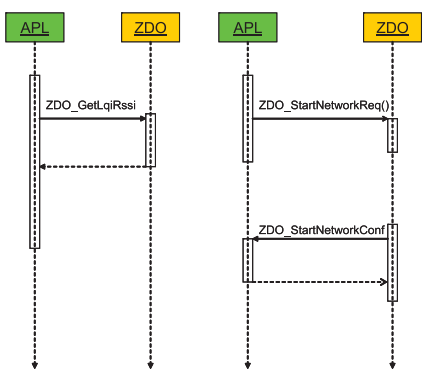

Рис. 2. Диаграммы последовательностей вызова функций: а) синхронного; б) асинхронного

В контексте стека протоколов BitCloud существует одно приложение, исполняемое на вершине стека. За ресурсы управляющего процессора борются не несколько приложений, а приложение и функции или сервисы стека протоколов. По сути, идет взаимодействие между приложением и сетевым стеком в рамках модели кооперативной многозадачности.

Следует обратить внимание на тот факт, что функции выполняются с приоритетом того уровня, к которому они принадлежат, например функция ***ZDO_StartNetworkConf*** выполняется с приоритетом уровня ZDO. Пользователь должен отслеживать и контролировать появление длинных командных последовательностей в программе, включая вложенные вызовы.

Для реализации вызовов длительностью более 10 мс предлагается использование так называемого отложенного исполнения. Отложенное исполнение предполагает переключение исполнения функции и стековых операций при помощи API-функций менеджера задач. Идея отложенного исполнения состоит в сохранении текущего состояния приложения и передаче/постановке задачи в очередь:

```
SYS_PostTask(APL_TASK_ID);
```

Операция постановки задачи в очередь является синхронной, и эффект от ее вызова состоит в уведомлении планировщика в том, что уровень имеет исполняющуюся задачу. ***APL_TaskHandler*** выполняется с приоритетом приложения — т. е. выполняется только тогда, когда более приоритетные задачи завершены. Это в свою очередь дает более длительное время выполнения задачи в контексте менеджера задач по сравнению с контекстом функций обратного вызова.

Совместное выполнение нескольких приложений предполагает существование нескольких независимых потоков или задач, выполняющихся одновременно. В BitCloud один контролирующий выполнение поток разделяется между прикладными задачами, сервисами и функциями стека протоколов.

Из-за непредсказуемости прерываний и возможного доступа разных функций к одним и тем же данным программист должен предусмотреть механизмы корректного доступа. Запуская задачу на определенном уровне стека, поток получает определенный этим уровнем приоритет, однако поток просто исполняет последовательность невзаимодействующих функций различных уровней стека и функций прикладной задачи. В общем случае выполнение отдельных функций обратного вызова не может быть взаимодействующим — каждая функция выполняется в своем цельном блоке кода.

Существуют условия, при которых выполнение текущей функции может прерваться. Это происходит при аппаратных прерываниях — возникновении событий на периферийных устройствах управляющего процессора (UART, SPI, IRQ и др.). Данная ситуация аналогично происходит и в любой другой системе.

Рис. 3 демонстрирует взаимодействие между приложением, стеком протоколов, менеджером задач и прерываниями. Изначально менеджер задач выполняет приложение, определяемое ***APL_TaskHandler***. Выполнение прерывается аппаратным событием (отображено серым). Обрабатывающая прерывание функция вызывается обработчиком или сервисом обслуживания прерываний. После завершения обработки прерывания управление возвращается приложению. После завершения работы менеджера задач управление передается планировщику, который выбирает для исполнения следующую задачу MAC-уровня. Пока выполняется ***MAC_TaskHandler***, он исполняет функцию обратного вызова ZDO-уровня, которая прерывается еще одним аппаратным прерыванием. Затем MAC-уровень выполняет еще одну ZDO-функцию, включающую вызов от приложения. Т. о. функция приложения исполняется с приоритетом MAC-уровня, или как если бы она имела идентификатор ***MAC_TASK_ID***.

Приложение BitCloud может зарегистрировать свою программу обработки аппаратного прерывания. Обычно это делается при добавлении к модулю нового устройства, для которого в стандартной SDK нет драйверов. Вызов для добавления своего обработчика имеет следующий вид:

```
HAL_RegisterIrq(uint8_t irqNumber,
HAL_irqMode_t irqMode, void(*)(void) f);
```

где ***irqNumber*** — идентификатор одной из доступных линий аппаратных прерываний; ***irqMode*** — определяет, когда прерывание будет обрабатываться (режим обработки); ***f*** — пользовательская функция обработки прерывания.

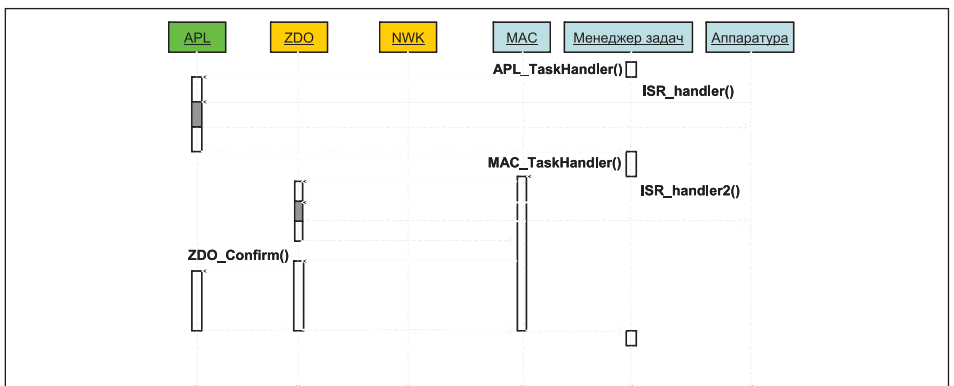

Рис. 3. Взаимодействие между приложением, стеком протоколов, менеджером задач и прерываниями

Выполнение обработчика прерывания может происходить одновременно с работой других приложений. При этом возможна ситуация, при которой будет запрошен доступ к глобальным состояниям одновременно как со стороны обработчика, так и со стороны приложения. В этом случае обработчику прерываний должен быть обеспечен исключительный доступ к такого рода данным, так как возможно возникновение «гонок», что может повлечь нарушение работы всего приложения.

Исключительный доступ обеспечивается макросами ***ATOMIC_SECTION_ENTER*** и ***ATOMIC_SECTION_LEAVE*** — они обеспечивают вход/выход в/из критической секции (блок кода, выполнение которого не нарушается прерываниями). Для того чтобы другие прерывания не были пропущены, длительность критической секции должна быть как можно меньше. На микроконтроллерах AVR используются флаги, указывающие на пропущенные прерывания, — т. о. прерывания по завершении критической секции будут обслужены.

Существуют три пути отслеживания выполнения приложения:

- через менеджер задач путем выполнения процедуры SYS_PostTask;
- через подтверждения функций обратного вызова по запросу;
- через асинхронные уведомления о событиях.

Ни один из этих методов не задает временных границ. Единственный путь убедиться в исполнении пользовательской процедуры через определенное время — использование таймеров.

Стек протоколов предоставляет интерфейс для доступа к таймерам, которые в свою очередь используют аппаратные таймеры управляющего процессора. Интерфейс доступа к таймерам является частью уровня HAL программного стека. Основная идея заключается в том, что структура ***HAL_AppTimer_t*** определяет интервал срабатывания таймера (в миллисекундах), тогда как сам таймер реализован как последовательность срабатываний более коротких таймеров или как несколько последовательно соединенных таймеров. Структура может быть передана функциям ***HAL_StartAppTimer*** или ***HAL_StopAppTimer*** для запуска или останова соответствующего таймера.

Принципы создания и функционирования сети ZigBee на основе стека BitCloud

Одним из самых интересных моментов в построении сетевых приложений распределенных сенсорных систем является проектирование и развертывание сети. Хотя сети стандарта ZigBee и относятся к самоорганизующимся, на некоторых моментах, связанных с их работой, следует остановиться. Безусловно, важными факторами при создании сети, помимо ее основных функций, являются количество узлов, требования к надежности, производительности, топологии. Поведение узла в сетях ZigBee зависит от его роли — координатор, маршрутизатор (в некоторых источниках — роутер, транзитный узел) или конечное устройство (функциональный узел) [7, 9, 10].

Для узла наиболее важны следующие моменты его работы: обнаружение и подключение к сети, обмен данными, выход из сети. Ниже эти моменты будут рассмотрены более подробно в контексте работы стека протоколов BitCloud. Все действия узла рассматриваются с точки зрения выполняемого сетевого приложения.

Процесс обнаружения и подключения к сети в стеке BitCloud обозначается термином «старт сети» или, более корректно, «инициализация сетевых операций». В рассматриваемом стеке протоколов нет разницы между процедурами создания новой или присоединения к уже существующей сети. Определена следующая процедура старта сети:

- задание параметров узла (тип узла, мощность передатчика, безопасность, управление энергопотреблением);
- указание специфических для сети параметров (задание схемы адресации, топология, параметры целевой сети);
- инициализация запроса старта сети (формирование запроса);
- получение подтверждения старта работы сети (подключение/не подключение).

Стек протоколов предусматривает широкий спектр конфигурационных параметров, которые определяют роль узла в сети и его характеристики. Управление этими настройками доступно через т. н. «сервер конфигурации» (ConfigurationServer, CS).

Все параметры, задаваемые сервером конфигурации, разделяются на две категории — постоянные и временные. Постоянные записываются в EEPROM управляющего контроллера, и их значения доступны приложениям и стеку после аппаратного сброса. Временные параметры заносятся в оперативную память, и, соответственно, могут быть потеряны при аппаратном сбросе (будут восстановлены их значения по умолчанию).

Файл ***configServer.h*** содержит определение всех параметров сервера конфигурации, используемые в любом из методов их задания: в ***Makefile*** приложения; в функции чтения/записи ***CS***.

Один из самых простых способов определения параметров узла — задание их в ***Makefile*** приложения. При этом значения, заданные по умолчанию в ***configServer.h,*** пропускаются и используются значения, описанные в ***Makefile***. Например, следующая строка задает выходную мощность 3 дБ:

```
+= -DCS_RF_TX_POWER=3.
```

Однако этот способ позволяет задавать параметры только во время компиляции приложения, а никак не в процессе его выполнения. Для изменения конфигурации «на лету» используются API-функции — ***CS_ReadParameter*** и ***CS_WriteParameter***. Каждая из них требует идентификатор параметра:

```
int8_t new_txPwr=3; // variable for writing new value
int8_t curr_txPwr; // variable for reading current value
CS_ReadParameter(CS_RF_TX_POWER_ID, &curr_txPwr);
CS_WriteParameter(CS_RF_TX_POWER_ID, &new_txPwr);
```

В BitCloud тип устройства определяется параметром ***CS_DEVICE_TYPE***, который может принимать следующие значения:

- DEVICE_TYPE_COORDINATOR (0x00) — координатор;
- DEVICE_TYPE_ROUTER (0x01) — маршрутизатор (роутер);
- DEVICE_TYPE_END_DEVICE (0x02) — конечное устройство.

Дополнительно параметр ***CS_RX_ON_WHEN_IDLE*** должен быть ***TRUE*** для координатора и роутера, тогда как для конечных устройств можно выставить ***FALSE*** для т. н. «непрямого» приема данных (по запросу, при выходе из спящего режима).

Пример конфигурирования узла как конечного устройства:

```
DeviceType_t deviceType = DEVICE_TYPE_END_DEVICE;
bool rxOnWhenIIde = false;
CS_WriteParameter(CS_DEVICE_TYPE_ID,&devicType);
```

Предусмотрено два уровня безопасности — без обеспечения защиты данных и стандартная сетевого уровня. В директории ***lib*** присутствует два набора библиотек — с поддержкой безопасности и без. Выбор производится на этапе компиляции приложения в ***Makefile*** установкой параметра ***SECURITY_MODE***.

Следует сказать, что поддержка безопасности требует больше памяти для хранения программ, что может вызвать ее дефицит (табл. 2). Возможно, придется корректировать значения параметров ***CS_NEIB_TABLE_SIZE, CS_ROUTE_TABLE_SIZE, CS_MAX_CHILDREN_AMOUNT***.

Как определено в стандарте ZigBee, каждое устройство может иметь уникальный 64-разрядный адрес, называемый также MAC или IEEE-адресом. В стеке BitCloud за него отвечает параметр ***CS_UID*** (на некоторых отладочных платформах, если параметр ***CS_UID*** установлен 0, MAC-адрес будет равен UID кристалла или взят из внешней EEPROM). При подключении к сети каждый узел идентифицируется т. н. «коротким» адресом (16 бит), иногда называемым также сетевым (NWK) адресом. Он используется в заголовках пакетов при обмене данными.

В BitCloud сетевой адрес может назначаться стеком (стохастическая адресация) или определяться пользовательским приложением (статическая адресация). Использование смешанной адресации в данном стеке не предполагается (настоятельно не рекомендуется). Способ адресации определяется параметром ***CS_NWK_UNIQUE_ADDR***: ***FALSE*** —стохастическая, ***TRUE*** — статическая. В последнем случае короткий адрес задается параметром ***CS_NWK_ADDR***. При этом координатор сети должен всегда иметь адрес 0x0000 (при стохастической адресации он присваивается координатору автоматически).

После старта сети устройство может узнать свой сетевой адрес из поля ***.shortAddr*** в аргументе процедуры ***ZDO_StartNetworkConf***.

При стохастической адресации работает специализированный механизм автоматического обнаружения конфликтов. В случае конфликта узел уведомляется функцией ***ZDO_MgmtNwkUpdateNotf()*** со статусом

Таблица 2. Оценка занимаемой стеком оперативной памяти

Параметр	Размещение записей	Требуемый размер памяти, незащищенный режим/режим безопасности, байт
CS_NEIB_TABLE_SIZE	в таблице соседних узлов	49 p*
CS_ROUTE_TABLE_SIZE	в таблице маршрутизации	7 p
CS_DUPLICATE_REJECTION_TABLE_SIZE	в таблице удаления дублирующихся записей	5 p
CS_ADDRESS_MAP_TABLE_SIZE	в карте адресов	11 p
CS_ROUTE_DISCOVERY_TABLE_SIZE	в таблице временных записей поиска маршрута	11 p
CS_NWK_DATA_REQ_BUFFER_SIZE	в NWK-буфере запросов на данные	47 p/56 p
CS_NWK_DATA_IND_BUFFER_SIZE	в NWK-буфере указателей на данные	168 p/182 p
CS_APS_DATA_REQ_BUFFER_SIZE	в ASP-буфере запросов на данные	38 p/150 p
CS_APS_ACK_FRAME_BUFFER_SIZE	в ASP-буфере подтверждения данных	35 p/64 p
CS_NWK_BTT_SIZE	в таблице передач широковещательных сообщений	4 p

Примечание. *p – количество соседних узлов

ZDO_NWK_UPDATE_STATUS (0x8E) с новым значением адреса в аргументе. В сетях со статической адресацией задача обнаружения конфликтов лежит на приложении. Однако в стеке предусмотрены некоторые функции, позволяющие это сделать — например вызовом функции ***ZDO_MgmtNwkUpdateNotf()*** со статусом ***ZDO_STATIC_ADDRESS_CONFLICT_STATUS (0x95)*** на узле, обнаружившем конфликт (может не совпадать с узлами с конфликтными адресами). Топология (организация) сети почти всегда отличается от возможных путей передачи данных по сети (рис. 4).

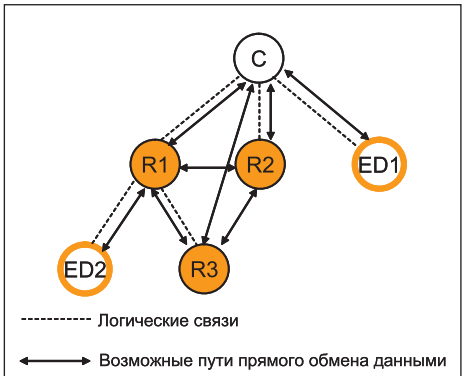

Рис. 4. Организационная структура сети и пути прямого обмена данными между узлами (глубина сети — 2, максимальное количество дочерних узлов — 2)

В добавление к описанным выше параметрам узлов существуют т. н. топологические параметры, влияющие на конфигурацию сети и способы присоединения новых узлов к ней, а также на функционирование и производительность сети в целом.

Для конечного устройства основным параметром является ***CS_NEIB_TABLE_SIZE***, определяющий максимальное число потенциальных родительских узлов, к которым данное устройство может попытаться подключиться при инициализации процедуры присоединения к сети. При обнаружении нескольких возможных родительских узлов выбирается тот, который имеет наилучшие показатели качества связи. Для маршрутизаторов этот параметр имеет дополнительный эффект — он ограничивает число узлов, которые могут быть выбраны в качестве родительских.

Также можно ограничивать количество подключаемых дочерних узлов — при помощи параметра ***CS_MAX_CHILDREN_AMOUNT,*** определяющего максимальное количество маршрутизаторов, подключенных к координатору или к маршрутизатору. Соответственно, разность ***CS_MAX_CHILDREN_AMOUNT*** и ***CS_MAX_ROUTER_CHILDREN_AMOUNT*** дает максимальное количество конечных устройств, подключаемых к данному узлу. На координаторе и маршрутизаторах должно выполняться правило: ***CS_NEIB_TABLE_SIZE > CS_MAX_CHILDREN_AMOUNT ≥ CS_MAX_ROUTER_CHILDREN***.

Максимальная глубина сети задается параметром ***CS_MAX_NETWORK_DEPTH***. Он определяет максимальное количество переходов (хопов) в сети от любого узла к координатору. Если максимальная глубина сети исчерпывается на заданном маршрутизаторе, он будет не в состоянии иметь подключенные/дочерние узлы, даже если его конфигурация это позволяет. Дополнительно параметр ***CS_MAX_NETWORK_DEPTH*** влияет на некоторые временные интервалы, используемые в стеке протоколов и гарантирующие корректную работу сетевых операций, — этот параметр должен быть одинаковым для всех узлов сети.

До выполнения процедуры старта узел должен задать параметры сети, которую он хочет создать, для координаторов, или к которой он хочет подключиться — для конечных устройств или маршрутизаторов:

- вид модуляции (***CS_CHANNEL_PAGE***) в соответствии со стандартом IEEE 802.15.4 (игнорируется для диапазона 2,4 ГГц, для диапазонов 868/915 МГц может иметь значения 0 или 2, для 780 МГц всегда имеет значение 5);
- частотный канал — 32-битное поле, определяющее частотные каналы, поддерживаемые узлом (через 32-битную маску — ***CS_CHANNEL_MASK*** — старшие 5 бит маски должны быть установлены в 0; остальные 27 определяют возможность узлом использовать канал: 1 — разрешено, 0 — запрещено);
- расширенный 64-битный идентификатор сети (***CS_EXT_PAN_ID***);
- параметры безопасности.

Если устройство желает подключиться к конкретной сети, оно должно установить свой ***PAN_ID*** таким же, как и у координатора желаемой сети. Если ***CS_EXT_PAN_ID*** узла (конечного устройства или маршрутизатора) равен 0, он может подключиться к первой же обнаруженной им сети.

Координатор при создании сети определяет короткий 16-битный идентификатор сети, который будет использован в заголовках пакетов. По умолчанию идентификатор сети генерируется случайным образом, и при обнаружении сети с аналогичным идентификатором координатор повторяет процедуру его генерации для исключения дальнейших конфликтов при передаче данных.

Такой механизм имеет непредсказуемый эффект — если координатор по какой-либо причине сбрасывается и заново инициализирует сеть с тем же PAN ID на том же канале, он может обнаружить маршрутизаторы и конечные узлы предыдущей сети и сформировать сеть с новым PAN ID. Однако часто требуется, чтобы координатор присоединил бы прежнюю сеть, для этого PAN ID определяется приложением перед стартом сети:

```
bool predefPANID=true;
uint16_t nwkPANID=0x1111;
CS_WriteParameter(CS_PREDEFINED_PAN_ID_ID,
&predefPANID);
CS_WriteParameter(CS_NWK_PAN_ID, &nwkPANID);
```

Если PAN ID задается на узле другого типа, этот узел имеет возможность войти в сеть только с установленными параметрами.

Если выбрана поддержка безопасности, для корректной работы библиотек конфигурируются дополнительно параметры ***CS_ZDO_SECURITY_STATUS*** и ***CS_NETWORK_KEY.*** При ***CS_ZDO_SECURITY_STATUS = 0*** все устройства сети должны иметь предварительно скомпилированный сетевой ключ (***CS_NETWORK_KEY***), определенный в ***Makefile***. ***CS_ZDO_SECURITY_STATUS= 3*** — одно из устройств (обычно координатор) работает как центр доверия — т. е. имеет установленный в ***Makefile*** ключ ***CS_NETWORK_KEY***. Все остальные устройства такого ключа на этапе компиляции не имеют и должны будут об-

ратиться по заданному параметром ***CS_APS_TRUST_CENTER_ADDRESS*** адресу.

Заключение

Стек протоколов BitCloud и основанные на его базе решения — средства разработки программного обеспечения и стандартные профайлы — предоставляют чрезвычайно мощный инструментарий для создания и поддержки разнообразных сетевых приложений. Поддерживаются различные топологии сетей, маршрутизация данных, различные схемы адресации, автоматизированы процедуры формирования сети. При необходимости возможно шифрование сетевого трафика. Полная поддержка стандартов обеспечивает совместимость с решениями на другой элементной базе, формализует процесс создания и сопровождения программного обеспечения.

Литература

1. ZigBit 2.4 GHz Wireless Modules Balanced RF Output and Dual Chip Antenna. http://www.atmel.com/dyn/resources/prod_documents/doc8226.pdf.
2. ZigBit 2.4 GHz Amplified Wireless Modules UFL-connector and Un-balanced RF Output. http://www.atmel.com/dyn/resources/prod_documents/doc8228.pdf.
3. ZigBit 700/800/900 MHz Wireless Modules Balanced RF Output. http://www.atmel.com/dyn/resources/prod_documents/doc8227.pdf.
4. RZRAVEN Firmware Documentation. http://www.atmel.com/forms/avr_raven_download.asp?fn=dl_AVR2017_RZRAVEN_Firmware.zip&family_id=676.
5. Сидоренко Б. Продукция компании Atmel для беспроводных сетей IEEE 802.15.4/ZigBee/6loWPAN // Электроника: НТБ. 2009. № 6. http://www.electronics.ru/pdf/6_2009/2065.pdf.
6. Atmel Products — MCU Wireless — Tools & Software. http://www.atmel.com/dyn/products/tools.asp?family_id=676.
7. Latest ZigBee Specification Including the PRO Feature Set. http://www.zigbee.org/Markets/ZigBeeSmartEnergy/Specification.aspx.
8. ZigBee RF4CE Specification. http://zigbee.org/Markets/ZigBeeRF4CE/download.aspx.
9. Скуснов А. ZigBee: обзор технологии // Компоненты и технологии. 2005. № 3.
10. Скуснов А. ZigBee: взгляд вглубь // Компоненты и технологии. 2005. № 4.
11. AVR2050: BitCloud User Guide. http://www.atmel.com/dyn/resources/prod_documents/doc8199.pdf.
12. AVR2052: BitCloud Quick Start Guide. http://www.atmel.com/dyn/resources/prod_documents/8200.pdf.